

**IN THE UNITED STATES PATENT AND TRADEMARK OFFICE
BEFORE THE BOARD OF PATENT APPEALS AND INTERFERENCES**

In re Application of:

Sunay Tripathi

Serial No.: 10/683,933

Filed: October 10, 2003

For: **A SYSTEM AND METHOD FOR
VERTICAL PERIMETER PROTECTION**

Confirmation No. 2203

Art Unit: 2143

Examiner: Jerry B. Dennison

Atty Docket No. SUN-P8978

Mail Stop Appeal Brief - Patents
Commissioner for Patents
P.O. Box 1450
Alexandria, VA 22313-1450

APPEAL BRIEF UNDER 37 CFR § 41.37

I. Real Party in Interest

Sun Microsystems, Inc.
4150 Network Circle
Santa Clara, CA 95054
USA

II. Related Appeals and Interferences

No other appeals or interferences are currently known to Appellants that will directly affect, be directly affected by, or have a bearing on the decision to be rendered by the Board of Patent Appeals and Interferences in the present appeal.

III. Status of Claims

Claims 1-8, 14, 19, 20, 23-28, 33, 36-39, 41-51, 53, 54, and 56-61 are pending in the application. Claims 9-13, 15-18, 21, 22, 29-32, 34, 35, 40, 52, and 55 have been cancelled. No claims have been allowed. The rejection of claims 1-8, 14, 19, 20, 23-28, 33, 36-39, 41-51, 53, 54, and 56-61 is the subject of this appeal.

IV. Status of Amendments

No claim amendments were filed after the final Office Action, which was mailed on October 24, 2008. All claim amendments have been entered, and pending claims 1-8, 14, 19, 20, 23-28, 33, 36-39, 41-51, 53, 54, and 56-61 are provided in the attached Claims Appendix.

V. Summary of Claimed Subject Matter

Claims 1, 20, 36, 44, and 53 are independent claims that are being appealed.

Claim 1 is directed toward “a method for processing packets through a plurality of protocol layers.” With reference to Appellants’ specification such a method may include all or some of the steps of the method 600 shown in Figure 6 and be carried out with the hardware, firmware, and/or software shown in Figures 1-4 with particular reference to the server 400 of Figure 4 that is shown to have multiple processors 410, 412, 414 that each use a queue 416, 418, 420 and local cache 404, 406, 408 to process packets through protocol layers. The server of Figure 4 is described in detail from line 6, page 17 to line 17, page 20. The method 600 of Figure 6 is described starting at line 5, page 22. Further, Figure 3 illustrates a communication connection 300 that shows protocol layers through which a packet may be processed and such processing is described beginning at line 15, page 15.

The method of claim 1 calls for “accessing a packet associated with a connection.” Again, Figure 3 shows an exemplary communication connection as called for in claim 1 (see, page 15, line 15), and the method 600 of Figure 6 begins with a step 602 regarding receiving a packet associated with a TCP connection and then examining its header information at step 604 and retrieving a connection structure ‘conn’ at step 606. These steps are described in more detail in lines 5-15 of page 22 of the Appellants’ specification.

The method of claim 1 then calls for “processing said packet through said plurality of protocol layers using a single thread from a single processor by assigning said connection to a single processor of a multiprocessor server system for processing.” Step 608 of method 600 calls for determining for the presence of a ‘conn’ entry and then at 610, the connection is assigned to a ‘squeue’ and a ‘conn_t’ is created. According to Appellants’ specification beginning at line 1 of

page 18, each processor (such as processor 410 of server 400 of Figure 4) has an associated 'squeue' (which is a serialization queue that is used for queuing packets destined for the associated processor). Further in lines 9-14 of page 18, it is noted that "a connection data structure called 'conn' that classifies each connection and provides routing information such that all packets associated with the connection are routed to a single assigned processor."

In the method of claim 1, "packets associated with said connection are directed to said single thread in said single processor for processing" and "connection state information used by said plurality of protocol layers is preserved by mutual exclusion of other threads from processing packets for said connection through said plurality of protocol layers" (e.g., see page 24, lines 1-6 that discuss that a thread processes a packet through the protocol layers without interruption and the squeue is marked as busy in part blocking other threads from processing packets through the connection). Step 612 of method 600 in Figure 6 calls for the examination of the conn_t data structure and routing packets for a connection to a particular/assigned squeue.

Lines 1-17 of page 20 described that a connection is assigned to a vertical perimeter instance attached to a particular processor such that an appropriate squeue can be retrieved. Then, "Vertical perimeters advantageously assure that only a single thread can process a given connection at any time, thus serializing access to the TCP connection structure by multiple threads...in merged TCP/IP module." And, further, "once a packet is being processed through the protocol layers by its processor, the thread is usually not interrupted unless to schedule a new task on its queue."

Independent claim 20 is directed toward a method for processing packets, with limitations similar to those found in claim 1, but the summary provided above is restated for the sake of completeness.

The method of claim 20 calls for "accessing a packet associated with a connection." Figure 3 shows an exemplary communication connection as called for in claim 1 (see, page 15, line 15), and the method 600 of Figure 6 begins with a step 602 regarding receiving a packet associated with a TCP connection and then examining its header information at step 604 and

retrieving a connection structure 'conn' at step 606. These steps are described in more detail in lines 5-15 of page 22 of the Appellants' specification.

The method of claim 20 then "assigning said packet to a processing queue associated with a single processor of a multi processor server system." Step 608 of method 600 calls for determining for the presence of a 'conn' entry and then at 610, the connection is assigned to a 'squeue' and a 'conn_t' is created. According to Appellants' specification beginning at line 1 of page 18, each processor (such as processor 410 of server 400 of Figure 4) has an associated 'squeue' (which is a serialization queue that is used for queuing packets destined for the associated processor). Further in lines 9-14 of page 18, it is noted that "a connection data structure called 'conn' that classifies each connection and provides routing information such that all packets associated with the connection are routed to a single assigned processor."

The method of claim 20 then indicates that "said processing queue provides uninterrupted single threaded processing of said data packet associated with the connection through a plurality of protocol layers using a single thread of the single processor" and the "state information of the packet within the connection is preserved so as to mutually exclude other threads from processing packets of said connection through said plurality of protocol layers" (e.g., see page 24, lines 1-6 that discuss that a thread processes a packet through the protocol layers without interruption and the squeue is marked as busy in part blocking other threads from processing packets through the connection). Step 612 of method 600 in Figure 6 calls for the examination of the conn_t data structure and routing packets for a connection to a particular/assigned squeue.

Lines 1-17 of page 20 described that a connection is assigned to a vertical perimeter instance attached to a particular processor such that an appropriate squeue can be retrieved. Then, "Vertical perimeters advantageously assure that only a single thread can process a given connection at any time, thus serializing access to the TCP connection structure by multiple threads...in merged TCP/IP module." And, further, "once a packet is being processed through the protocol layers by its processor, the thread is usually not interrupted unless to schedule a new task on its queue."

Independent claim 36 is directed to a multiprocessor server system with limitations similar to those called for in claims 1 and 20. The server system may take the form of server system 400 shown in Figure 4 and may also act to perform all or some of the steps of the method 600 shown in Figure 6 and be carried out with the hardware, firmware, and/or software shown in Figures 1-3. Claim 36 calls for a number “processors for processing packets through a plurality protocol layers, and these may be found in the server 400 of Figure 4 with processors 410, 412, 414 that each use a queue 416, 418, 420 and local cache 404, 406, 408 to process packets through protocol layers. The server of Figure 4 is described in detail from line 6, page 17 to line 17, page 20 while the method 600 of Figure 6 is described starting at line 5, page 22. Further, Figure 3 illustrates a communication connection 300 that shows protocol layers through which a packet may be processed and such processing is described beginning at line 15, page 15.

Claim 36 further calls for “a plurality of threads running in the plurality of processors.” Page 4, lines 2-15 describe the processors (such as processors 410, 412, 414 each using a queue along with a worker thread controlled by queue and “both are bound to a single processor” within the multiple processor server system (such as system 400 of Figure 4). Further, claim 36 calls for “a plurality of queues, each queue associated with a respective processor of said plurality of processors.” These may be the queues 416, 418, 420 shown in Figure 4 and associated with one of the processors 410, 412, 414, respectively, and these queues are defined and described at least in lines 9-16 of page 5 and a representative queue data structure is shown in Table 1 of page 17.

Claim 36 further calls for “a memory resident connection data structure for assigning packets of a connection to a queue of said plurality of queues for processing said packets of said connection using a single thread associated with the single queue of a corresponding processor of said plurality of processors.” Figure 4 shows a “Conn_t” data structure 402 in the server system 400. Page 19, lines 4-21 describes use of a lookup for inbound packets using a connection data structure that is retrieved from connection classifier hash table (conn_t).

Independent claim 44 is directed to a computer system that includes a processor that executes instructions for processing data packets. Again, this may be a server as shown at 400 in

Figure 4 and may operate as shown in methods 600 and 700 of Figures 6 and 7 to provide the claimed steps/functions. The method performed by processor includes “accessing a packet associated with a connection.” Figure 3 shows an exemplary communication connection as called for in claim 1 (see, page 15, line 15), and the method 600 of Figure 6 begins with a step 602 regarding receiving a packet associated with a TCP connection and then examining its header information at step 604 and retrieving a connection structure ‘conn’ at step 606. These steps are described in more detail in lines 5-15 of page 22 of the Appellants’ specification.

The processor also executes instructions to perform “processing said packet through said plurality of protocol layers using a single thread from a single processor by assigning said connection for processing to a single processor of a multiprocessor server system wherein packets associated with said connection are directed to said single thread associated with said single processor for processing, wherein connection state information used by said plurality of protocol layers is preserved by mutual exclusion of other threads processing packets for said connection through said plurality of protocol layers.” See, for example, page 24, lines 1-6 that discuss that a thread processes a packet through the protocol layers without interruption and the queue is marked as busy in part blocking other threads from processing packets through the connection). Step 612 of method 600 in Figure 6 calls for the examination of the conn_t data structure and routing packets for a connection to a particular/assigned queue.

Also, lines 1-17 of page 20 described that a connection is assigned to a vertical perimeter instance attached to a particular processor such that an appropriate queue can be retrieved. Then, “Vertical perimeters advantageously assure that only a single thread can process a given connection at any time, thus serializing access to the TCP connection structure by multiple threads...in merged TCP/IP module.” And, further, “once a packet is being processed through the protocol layers by its processor, the thread is usually not interrupted unless to schedule a new task on its queue.”

Independent claim 53 is directed to a computer system with a processor for processing data packets including executing instructions to perform method steps similar to those found in claim 20. Again, the system may be the server system 400 of Figure 4 that is described

beginning in detail beginning at line 6 of page 17. The method performed includes “accessing a packet associated with a connection.” Figure 3 shows an exemplary communication connection as called for in claim 1 (see, page 15, line 15), and the method 600 of Figure 6 begins with a step 602 regarding receiving a packet associated with a TCP connection and then examining its header information at step 604 and retrieving a connection structure ‘conn’ at step 606. These steps are described in more detail in lines 5-15 of page 22 of the Appellants’ specification.

The computer system of claim 53 also performs “assigning said packet to a processing queue associated with a single processor of a multi processor server system, wherein said processing queue provides uninterrupted single threaded processing of said data packet associated with the connection through a plurality of protocol layers using a single thread of the single processor, wherein state information of the packet within the connection is preserved so as to mutually exclude other threads from processing packets of said connection through said plurality of protocol layers.” Step 608 of method 600 calls for determining for the presence of a ‘conn’ entry and then at 610, the connection is assigned to a ‘squeue’ and a ‘conn_t’ is created. According to Appellants’ specification beginning at line 1 of page 18, each processor (such as processor 410 of server 400 of Figure 4) has an associated ‘squeue’ (which is a serialization queue that is used for queuing packets destined for the associated processor). Further in lines 9-14 of page 18, it is noted that “a connection data structure called ‘conn’ that classifies each connection and provides routing information such that all packets associated with the connection are routed to a single assigned processor.”

See also page 24, lines 1-6 that discusses that a thread processes a packet through the protocol layers without interruption and the squeue is marked as busy in part blocking other threads from processing packets through the connection). Step 612 of method 600 in Figure 6 calls for the examination of the conn_t data structure and routing packets for a connection to a particular/assigned squeue. Lines 1-17 of page 20 described that a connection is assigned to a vertical perimeter instance attached to a particular processor such that an appropriate squeue can be retrieved. Then, “Vertical perimeters advantageously assure that only a single thread can process a given connection at any time, thus serializing access to the TCP connection structure by multiple threads...in merged TCP/IP module.”

VI. Grounds of Rejection to be Reviewed on Appeal

Claims 1-4, 20, 27, 28, 33, 36, 37, 39, 41-47, 53, 54, 60, and 61 stand rejected under 35 U.S.C. §102(e) as being anticipated by U.S. Pat. No. 7,076,545 (“DiMambro”). The rejection of claims 1-4, 20, 27, 28, 33, 36, 37, 39, 41-47, 53, 54, 60, and 61 as being anticipated by DiMambro is the first of the grounds of rejection that is the subject matter of this appeal.

Claims 1-5, 20, 27, 28, 44-48, 53, 54, 60, and 61 stand rejected under 35 U.S.C. §102(b) as being anticipated by U.S. 6,338,078 (“Chang”). The rejection of claims 1-5, 20, 27, 28, 44-48, 53, 54, 60, and 61 as being anticipated by Chang is the second grounds of rejection that is the subject matter of this appeal.

Claim 14 stands rejected under 35 U.S.C. §103(a) as being unpatentable over Chang, and this rejection is the third grounds of rejection that is the subject matter of this appeal.

Claims 6-8, 19, 23-26, 33, 36-39, 41-43, 49-51, and 56-59 stand rejected under 35 U.S.C. §103(a) as being unpatentable over Chang in view of U.S. Pat. Appl. Publ. No. 2002/0112188 (“Syvanne”). The rejection of claims 6-8, 19, 23-26, 33, 36-39, 41-43, 49-51, and 56-59 as being obvious in light of Chang and Syvanne is the fourth grounds of rejection that is the subject matter of this appeal.

VII. Arguments

Rejection of Claims 1-4, 20, 27, 28, 33, 36, 37, 39, 41-47, 53, 54, 60, and 61 Under 35 U.S.C. §102 Based on DiMambro Improper

In the final Office Action mailed October 24, 2008 (the “final Office Action”), claims 1-4, 20, 27, 28, 33, 36, 37, 39, 41-47, 53, 54, 60, and 61 were rejected under 35 U.S.C. §102(e) as being anticipated by DiMambro. Appellants request that the rejection be reversed because DiMambro fails to teach or suggest at least one limitation in independent claims 1, 20, 36, 44, and 53.

In response to the final Office Action, Appellants appealed the rejection of all pending claims such that there was no Advisory Action issued. The final Office Action retained the

rejections presented in the prior Office Action and provided the Examiner's response to certain ones of the Appellants' arguments/reasons for allowing the pending claims.

Prior to looking at specific claim language, it may be useful to briefly discuss aspects of Appellants' invention. Appellants' Background explains that with conventional computer systems a layered approach is used in network communications in which each protocol layer and the use of multi-CPU environment causes processing of "the same packets on more than on CPU through the software layers (protocol layers)" that "leads to excessive context switching, increased latencies, and poor CPU data locality" (e.g., protocol data is not stored locally. Further, as a result, "packets for the same connection go through various protocol layers where they have to contend for access to their state structures at each layer."

In the Summary, Appellants explain that their solution involves packets for a particular connect to go through all the protocol layers while providing mutual exclusion for its state structure at each protocol layer, and "a communication packet of a given connection is processed from beginning to end by a single processor without contending for additional locks and getting queued at each protocol layer. To implement this vertical perimeter framework, a kernel data structure called a "queue" (or serialization queue type") and a worker thread that is controlled by the queue, and both the queue and the single thread are bound to a single processor. At the bottom of page 4 of Appellants' specification, it is noted that a queue "is processed by a single thread at a time and all data structures used to process a given connection from within the perimeter can be accessed without additional locking or mutual exclusion, thus improving both processor, e.g., CPU, performance and thread context data locality. Access of the connection meta data, the packet meta data, and the packet payload data is localized, thus reducing retrieval time for such data (e.g., in a localized cache specific to the CPU processing the packet)."

Turning specifically to claim language, claim 1 is directed toward a method for processing packets through a number of protocol layers. The method includes "accessing a packet associated with a connection." A connection in this sense is described in the specification beginning at line 20 of page 13 as being, for example, a TCP connection that provide full duplex data transfer and then with reference to Figure 3 as being a communication connection 300 that

includes multiple protocol layers (e.g., the application layer, transport, and network layers of the TCP/IP protocol), and “the connection state for individual modules of connection 300 may be incorporated into a single connection structure called ‘conn’. With reference to Figure 4 and page 19, line 9, the connection data structure or ‘conn’ is used to provide a lookup for inbound packets, and it’s use and processing causes “all packets for the same connection” to be processed on the queue to which the connection is bound (e.g., this causes processing of all packets associated with “a connection” as called for in claim 1 to be performed by a single processor to reduce data state conflicts between protocol layers and a localized cache can further decrease processing time).

With this conn data structure in mind, claim 1 also calls for processing said packet through said plurality of protocol layers using a single thread from a single processor “by assigning said connection to a single processor of a multiprocessor server system for processing.” None of the references cited by the Examiner teach assigning “a connection to a single processor,” and claim 1 further requires that “wherein packets associated with said connection are directed to said single thread in said single processor for processing.” Hence, to teach or suggest the method of claim 1, the reference must show or suggest that all packets associated with a particular connection are sent to a single thread in a single processor, which would require maintenance of a data structure similar to ‘conn’ shown in Figure 4 such that inbound packets could be accessed to determine which thread/processor to assign them to. Neither anticipation-type reference provides such teaching.

Further, claim 1 builds on such a requirement for a conn-like data structure by stating “wherein connection state information used by said plurality of protocol layers is preserved by mutual exclusion of other threads from processing packets for said connection.” Therefore, a proper anticipation reference would also have to teach that connection state information is maintained for a connection (e.g., protocol layer state information maintained in a local cache or the like) and that other threads are mutually excluded, and Appellants’ Background makes it clear that the standard practice was that threads were not mutually excluded from such processing, and the references fail to show such mutual exclusion to accessing/processing a packet being processed upon by other threads.

The final Office Action cites DiMambro as showing the “accessing a packet associated with a connection” at col. 1, lines 39-40 and also then showing that all packets associated with this same connection are directed to a single thread of a single processor as required in claim 1 at col. 1, lines 41-50 and col. 3, lines 3-15. Appellants strongly disagree. In col. 1, lines 41-50 and col. 3, lines 3-15, DiMambro does make a brief reference to selection of which service queue on the basis of “communication connections.” However, this is only mentioned in passing and there is no indication that a single thread shall handle all processing of the packet and all packets associated with the connection (e.g., selection of a service queue based on a connection does not necessitate or even suggest that all packets of that connection will be processed by a single thread of a particular processor). To understand what is intended/meant by DiMambro, the entire teaching must be considered and not just the use of the word “connection” in isolation.

Specifically, as discussed in Appellants’ Pre-Appeal Brief Request for Review, DiMambro discloses a method for load balancing the processing of received packets, which may conflict directly with or at least solve a differing problem than the one addressed by Appellants that can be solved by using a single thread to handle packets of a connection through the various protocol layers. For example, DiMambro appears to teach away from the method of claim 1 in its first paragraph in col. 1 where it states that it provides methods for “distributing a portion of the processing of received packets among a plurality of threads” and not that it assigns all processing of received packets for a connection to a single thread. Likewise, at the top of col. 3, DiMambro teaches that “protocol processing can be load-balanced” to solve the problem being addressed, and Appellants teach away from load balancing of protocol processing by requiring a single processor to perform all the protocol layer processing for each packet associated with a connection. In col. 4, lines 29-39, the “ISR” is described as selecting one of a plurality of service queues for an inbound packet with selection made to send packets through a particular queue that are serviced by a plurality of threads (not just a single thread that is bound to the queue) to be “distributed or load balanced.” As discussed in col. 2, lines 58-65, the retrieval of a packet from a descriptor ring is “decoupled from the subsequent protocol related processing of the packet,” which teaches away from the concept of storing state information from protocol layers for use through the processing of a communication connection and its associated packets.

As discussed in the Pre-Appeal Brief Request for Review, DiMambro calls for packets to be received at a communication interface of a computer system and placed on a receive descriptor ring. An ISR (Interrupt Service Routine) or similar process retrieves packets from the ring and places the packet in one of a plurality of service queues that are tended by service threads. DiMambro further teaches that retrieval of a packet from the descriptor ring is decoupled from the subsequent protocol related processing of the packet. This allows the ISR to handle packets in shorter periods of time so that more packets can be handled in any given period of time. DiMambro asserts that by using multiple service queues and service threads to perform subsequent processing of the packets, efficient overall handling of received packets is enabled (see, DiMambro, col. 2, lines 35-57). From this discussion and understanding of DiMambro, it can be seen that this reference fails to show that all packets of a connection are “are directed to said single thread in said single processor for processing” as called for in the second limitation of claim 1. Instead, the teachings of DiMambro are all directed toward efficient load balancing during the processing received packets and using multiple queues and threads to process packets (rather than having packets forwarded to particular threads even if this would unbalance the overall packet processing loads – but would maintain packets of a particular connection with a particular thread and processor). For at least these reasons, DiMambro fails to anticipate claim 1, and Appellants request that the rejection be overturned or reversed by the Board.

Further, DiMambro fails to teach the limitation that “connection state information used by said plurality of protocol layers is preserved by mutual exclusion of other threads from processing packets for said connection through said plurality of protocol layers.” For example, the claimed invention may preserve the connection state information by utilizing localized cache so that the connection state information to process connection requests is maintained locally so that data state conflicts between protocol layers may be reduced.

The final Office Action cites DiMambro as teaching this limitation at col. 1, lines 41-50 and col. 3, lines 3-15. In the cited portion of col. 1, there is no reference whatsoever that “connection state information” is preserved (such as in a local cache or the like). There is also no mention that mutual exclusion of other threads is provided in DiMambro, and it should be remembered that the more standard method is that discussed in Appellants’ Background that

more than one process or thread may perform protocol layer processing for a single packet. In col. 3 at the cited lines there is also no mention of preserving connection state information or use of mutual exclusion of other threads, and there is no mention of how such connection state information may be used (e.g., DiMambro provides a different process involving load balancing and there would be no/less reason for maintaining such state information for a connection). Of course, an anticipation rejection requires that a reference teach each and every limitation of a claim, and Appellants urge the Board to reverse the anticipation rejection of claim 1 because DiMambro fails to show the limitations of claim 1 involving preserving connection state information for protocol layers via mutual exclusion of other threads from processing the connection packets (i.e., there is no figure or text mentioning storing such state information or later using such information).

Claims 2-4 depend from claim 1 and are believed allowable over DiMambro at least for the reasons provided above for allowing claim 1 over this reference.

Independent claim 20 is directed toward a method of processing packets that includes limitations similar to those found in claim 1 (including the preserving of state information for a connection). Hence, the reasons for allowing claim 1 over DiMambro are believed applicable to claim 20. Claims 27, 28, and 33 depend from claim 20 and are believed allowable over DiMambro at least for the reasons provided for allowing claim 20 over this reference.

Independent claim 36 is directed toward a multiprocessor server system with limitations similar to those found in claims 1 and 20 but in systems form. Hence, the reasons for allowing claim 1 over DiMambro are also believed applicable to claim 36. Claims 37, 39, 41-43 depend from claim 36 and are believed allowable over DiMambro at least for the reasons provided for allowing claim 36 over this reference.

Similarly, independent claims 44 and 53 are directed toward computer systems executing code instructions with limitations similar to those found in claims 1 and 20, and the reasons for allowing claim 1 over DiMambro are believed equally applicable to claims 44 and 53. Claims 45-47, 54, 60, and 61 depend from claim 44 or claim 53 and are believed allowable over DiMambro at least for the reasons provided for allowing claims 44 and 53.

Rejection of Claims 1-5, 20, 27, 28, 44-48, 53, 54, 60, and 61 Under 35 U.S.C. §102 Based on Chang is Improper

In the final Office Action, claims 1-5, 20, 27, 28, 44-48, 53, 54, 60, and 61 were rejected under 35 U.S.C. §102(b) as being anticipated by Chang. Appellants request that the rejection be reversed because the teaching of the Chang reference fails to teach or suggest at least one limitation in claim each of the pending independent claims and also fails to show features of the separately argued dependent claims discussed below. The summary of aspects of Appellants' invention and its use in construing the pending claims is provided above with regard to the arguments for allowing the pending claims over DiMambro.

As discussed above (but repeated here for completeness), claim 1 is directed toward a method for processing packets through a number of protocol layers. The method includes "accessing a packet associated with a connection." A connection in this sense is described in the specification beginning at line 20 of page 13 as being, for example, a TCP connection that provide full duplex data transfer and then with reference to Figure 3 as being a communication connection 300 that includes multiple protocol layers (e.g., the application layer, transport, and network layers of the TCP/IP protocol), and "the connection state for individual modules of connection 300 may be incorporated into a single connection structure called 'conn'. With reference to Figure 4 and page 19, line 9, the connection data structure or 'conn' is used to provide a lookup for inbound packets, and it's use and processing causes "all packets for the same connection" to be processed on the queue to which the connection is bound (e.g., this causes processing of all packets associated with "a connection" as called for in claim 1 to be performed by a single processor to reduce data state conflicts between protocol layers and a localized cache can further decrease processing time).

With this conn data structure in mind, claim 1 also calls for processing said packet through said plurality of protocol layers using a single thread from a single processor "by assigning said connection to a single processor of a multiprocessor server system for processing." None of the references cited by the Examiner teach assigning "a connection to a single processor," and claim 1 further requires that "wherein packets associated with said connection are directed to said single thread in said single processor for processing." Hence, to

teach or suggest the method of claim 1, the reference must show or suggest that all packets associated with a particular connection are sent to a single thread in a single processor, which would require maintenance of a data structure similar to 'conn' shown in Figure 4 such that inbound packets could be accessed to determine which thread/processor to assign them to. Neither anticipation-type reference provides such teaching.

Further, claim 1 builds on such a requirement for a conn-like data structure by stating "wherein connection state information used by said plurality of protocol layers is preserved by mutual exclusion of other threads from processing packets for said connection." Therefore, a proper anticipation reference would also have to teach that connection state information is maintained for a connection (e.g., protocol layer state information maintained in a local cache or the like) and that other threads are mutually excluded, and Appellants' Background makes it clear that the standard practice was that threads were not mutually excluded from such processing, and the references fail to show such mutual exclusion to accessing/processing a packet being processed upon by other threads.

Generally, with reference to Chang, this reference teaches a method of distributing input processing to multiple CPUs on a multiprocessor system to improve network throughput and to take advantage of multiprocessor scalability. According to Chang, packets are received from the network and distributed to N high priority threads, with "N" being the number of CPUs on the system. N queues are then provided to which the incoming packets are distributed. When one of the queues is started, as discussed in the Abstract and elsewhere in Change, one of the threads is scheduled to process packets on this queue at any one of the CPUs that happens to be presently available at the time (i.e., not because the processor and/or thread is bound to a queue or queue that is provided for a particular connection). Significantly, the IP queue is concurrently processed by multiple threads with one thread per CPU and one queue per thread (but again this is not per connection and not with state information for protocols preserved by exclusion). Chang, as discussed in col. 5, lines 21-26 is attempting to shorten path length with more CPUs running parts of packet's code simultaneously in a multithreaded fashion. In other words, Change teaches improving network throughput and using multiple threads to process packet of a

single connect – which is indirect contrast to the language of claim 1 and the other pending independent claims.

Specifically, claim 1 discusses accessing a packet for “a connection” and later assigning the connection to a single thread of a single processor” to process “packets associated with said connection.” Chang does not suggest or teach directing all packets of a connection to a single thread in a single processor for processing. In fact, Chang describes distributed input processing – versus directed processing for a connection – by assigning different packets to different queues even if they are associated with a single communication connection. Chang merely mentions processing a packet with a thread but does not suggest or teach the specific limitations of claim 1 (that all packets of a connection are processed by a single thread and its associated processor). This is significant in Appellants’ claimed method because by directing all packets of a connection to a single queue on a single processor, processing time is decreased by reducing data state conflicts between protocol layers (see, for example, page 19, lines 16-21 of Appellants’ specification).

In the final Office Action on page 15, there is no discussion by the Examiner that any of the cited portions teach accessing inbound packets and assigning all for a particular connection to a single thread of a single processor. Instead, the Examiner admits that his understanding of Chang and the cited portions was that “Packets are distributed by a hashing function, hashing addresses of the packets to determine which queue/CPU pair it should be sent to.” This is useful for distributing load or balancing load – but actually teaches away from the limitation of claim 1 as it will not result in all packets of a connection going to a bound/paired thread and processor for processing. For example, the Examiner cites Figure 3 for showing this limitation of claim 1, but it shows that a hash 50 determines which queue 62-68 a packet is passed – and there is no suggestion that the hash 50 acts to access a packet and determine which connection it belongs to and send it to a queue for that connect (e.g., where does Chang teach the queues 62-68 teach the queues are bound to a connection?).

The final Office Action cites to col. 2, line 64 to col. 3, line 2 as rebutting this argument or understanding of Chang, but Appellants disagree as this is a brief summary that must be

understood in light of the full Chang specification. In col. 2, lines 64-67, Chang states that the packets “are distributed to one of the N queues by using a hashing function based on the source MAC addresses, source IP address, or the packet’s source and destination TCP port number, or all or a combination of the foregoing.” Then, at col. 3, lines 1-2, Chang states that the “hashing mechanism ensures that the sequence of packets within a given communication session will be preserved.”

This, however, means that hashing may be provided in a variety of manners for packets to distribute load of network packets, but none of these methods show that the hash function assigns all packets for a connection to a particular queue or thread/processor. The order of the packets is maintained even though the packets are distributed across a variety of hashes by the hashing mechanism. For example, in col. 6, lines 47-50, Chang states “no matter which CPU 54-60 handles a particular queue, the packets associated with a particular device will flow to one of the applications 52-56 in sequence – but this statement does not call for a particular queue to be associated with a particular communication connection just that sequence is maintained in the procedure. The Board is asked to see Chang at col. 7, lines 40-47 where Chang summarizes that how a queue is selected – and to note that there is simply no mention of using a connection as the selection criteria – as instead queuing and parallelization is employed using a MAC address, then an IP address, and then the port addresses alone or in combination that are processing via the hashing shown in Figure 3. Based on all the foregoing, Chang fails to teach assigning all packets for a connection to a single thread on a processor for processing, and Chang cannot be said to anticipate the method of claim 1.

Further, Chang fails to teach maintaining “connection state information used by said plurality of protocol layers is preserved by mutual exclusion of other threads from processing packets for said connection.” For example, such connection state information may be maintained within a local cache associated the CPU bound to the connection such that this locally maintained information is useful for reducing data state conflicts between protocol layers. The Examiner cites Chang at Figure 3, queues 62-68, processors 54-60 and col. 2 lines 55-60 and col. 2, lines 64-67 as showing this limitation. However, these portions of Chang merely teach engaging a hashing function to determine which queue/CPU pair each inbound packet

should be sent to for processing with no mention or suggestion that connection state information is preserved or that such preservation is provided by mutual exclusion techniques. As different packets of a connection request may be routed and processed by different processors according to Chang (see col. 6, lines 30-32), any connection state information for a processor is different and each of the packets are packaged with the related processors' distinct connection information.

There is no mention in Chang of the packets being processed by a single processor with the connection state information of the specific processor (for example, to enable faster processing of packets of a connection). The Response to Arguments in the final Office Action is silent with regard to this argument (i.e., fails to rebut or address this separate reason for allowance of claim 1 over Chang). Appellants, therefore, request that a specific citation to Chang be provided that shows where or how state connection information is preserved/cached for a processor via mutual exclusion or that claim 1 be allowed over Chang because the reference fails to anticipate this limitation.

Claims 2-5 depend from claim 1 and are believed allowable over Chang at least for the reasons provided above for allowing claim 1 over this reference.

Independent claim 20 is directed toward a method of processing packets that includes limitations similar to those found in claim 1 (including the preserving of state information for a connection). Hence, the reasons for allowing claim 1 over Chang are believed applicable to claim 20. Claims 27 and 28 depend from claim 20 and are believed allowable over Chang at least for the reasons provided for allowing claim 20 over this reference.

Similarly, independent claims 44 and 53 are directed toward computer systems executing code instructions with limitations similar to those found in claims 1 and 20, and the reasons for allowing claim 1 over Chang are believed equally applicable to claims 44 and 53. Claims 45-48, 54, 60, and 61 depend from claim 44 or claim 53 and are believed allowable over Chang at least for the reasons provided for allowing claims 44 and 53.

Rejection of Claim 14 Under 35 U.S.C. §103 Based on Chang is Improper

In the final Office Action, claim 14 was rejected under 35 U.S.C. §103(a) as being unpatentable over Chang. Appellants request that the rejection be reversed because the teaching of Chang fails to suggest at least one limitation in claim 1, and claim 14 depends from claim 1.

Rejection of Claims 6-8, 19, 23-26, 33, 36-39, 41-43, 49-51, and 56-59 Under 35 U.S.C. §103 Based on Chang is Improper

In the final Office Action, claims 6-8, 19, 23-26, 33, 36-39, 41-43, 49-51, and 56-59 were rejected under 35 U.S.C. §103(a) as being unpatentable over Chang in view of Syvanne. Initially, independent claim 36 includes limitations similar to those found in claims 1 and 20, and it is believed allowable over Chang and Syvanne for the reasons provided for allowing claims 1 and 20 over these references.

Further, Appellants request that the rejection be reversed because the combined teaching of Chang and Syvanne fails to suggest at least one limitation in independent claims 1, 20, 44, and 53, from which these claims depend (and for reasons provided below for separate allowance of the dependent claims). Specifically, Syvanne fails to overcome the deficiencies of Chang discussed above with reference to the pending independent claims.


Appellants still further would like to point out that the Examiner on page 21 of the final Office Action admits that Chang fails to teach “generating a unique connection data structure specific to said connection.” Appellants argue this is because Chang simply does not maintain a queue that is bound to a particular connection and assign packets to a particular queue based on such accessing of a packet as required in the independent claims. In other words, if Chang provided the teaching asserted by the Examiner it would provide the explicit teaching for which the Examiner admits it is lacking (i.e., a reference that actually did teach assigning packets for a connection to a queue (e.g., an squeue) would teach creating a conn or similar data structure and would not be completely silent on such a significant aspect and the lack of any even passing reference indicates that this reference is not using such a structure because it does not need its functionality).

Conclusion

In view of the above remarks, the pending claims are believed allowable and the case in condition for allowance. Appellants respectfully request that the rejections of all pending claims be reversed. Appellants have met or exceeded the burden of overcoming the prima facie case of unpatentability made out by the Examiner by providing rebuttal evidence of adequate weight and explaining how and why the record indicates the pending claims are patentable.

Respectfully submitted,

Date: September 30, 2009


Kent A. Lembke, Reg. No. 44,866
Marsh Fischmann & Breyfogle LLP
8055 East Tufts Avenue, Suite 450
Denver, Colorado 80237
Phone: (720) 562-5507

VIII. CLAIMS APPENDIX

1. A method for processing packets through a plurality of protocol layers comprising:
 - accessing a packet associated with a connection; and
 - processing said packet through said plurality of protocol layers using a single thread from a single processor by assigning said connection to a single processor of a multiprocessor server system for processing wherein packets associated with said connection are directed to said single thread in said single processor for processing and wherein connection state information used by said plurality of protocol layers is preserved by mutual exclusion of other threads from processing packets for said connection through said plurality of protocol layers.
2. The method as described in claim 1 wherein said single thread is uninterrupted while processing said packet through said plurality of protocol layers.
3. The method as described in claim 1 further comprising assigning said packet to a processing queue wherein said processing queue provides single threaded processing of said packet through said plurality of protocol layers.
4. The method as described in claim 3 wherein said processing queue provides single threaded processing of said packet through said plurality of protocol layers by assigning only one packet to be processed by said plurality of protocol layers at a time.
5. The method as described in claim 4 wherein said packet is assigned to said processing queue based on address information of said connection.

6. The method as described in claim 1 further comprising generating a unique connection data structure specific to said connection based on address information of said connection stored in said packet associated with the connection.

7. The method as described in claim 6 wherein said address information comprises a local IP address and a remote IP address.

8. The method as described in claim 7 wherein said address information further comprises a remote port address and a local port address.

14. The method as described in claim 3 wherein said processing queue is an queue.

19. The method as described in claim 6 wherein subsequent data packets of said connection, are assigned to said single processor based on said connection data structure.

20. A method for processing packets comprising:
accessing a packet associated with a connection; and
assigning said packet to a processing queue associated with a single processor of a multi processor server system, wherein said processing queue provides uninterrupted single threaded processing of said data packet associated with the connection through a plurality of protocol layers using a single thread of the single processor, wherein state information of the packet within the connection is preserved so as to mutually exclude other threads from processing packets of said connection through said plurality of protocol layers.

23. The method as described in claim 20 further comprising generating a unique connection data structure associated with said connection based on address information of said connection stored in said packets associated with the connection.

24. The method as described in claim 23 wherein said address information comprises a local IP address and a remote IP address.

25. The method as described in claim 24 wherein said address information further comprises a remote port address and a local port address.

26. The method as described in claim 25 wherein said connection data structure is used to assign subsequent packets associated with said connection to said processing queue.

27. The method as described in claim 20 wherein said plurality of protocol layers includes a TCP protocol layer.

28. The method as described in claim 20 wherein said plurality of protocol layers includes an IP protocol layer.

33. A method as described in claim 23 wherein said queue is associated with said connection data structure.

36. A multiprocessor server system comprising:
a plurality of processors for processing packets through a plurality of protocol layers;

a plurality of threads running in the plurality of processors;

a plurality of queues, each queue associated with a respective processor of said plurality of processors; and

a memory resident connection data structure for assigning packets of a connection to a queue of said plurality of queues for processing said packets of said connection using a single thread associated with the single queue of a corresponding processor of said plurality of processors.

37. A multiprocessor server system as described in claim 36 wherein said connections are TCP connections.

38. A multiprocessor server system as described in claim 37 wherein said plurality of protocol layers comprise: IP; TCP; and socket layers.

39. A multiprocessor server system as described in claim 36 wherein a processor of said plurality of processors processes a packet of its queue without interruption through said plurality of protocol layers except for scheduling another packet on its queue.

41. A multiprocessor server system as described in claim 37 wherein said connection data structure is established for a new connection upon receiving a new connection request and wherein said connection data structure comprises an identifier of a queue associated with the single thread of the same corresponding processor to which all packets of said new connection are to be assigned.

42. A multiprocessor server system as described in claim 36 further comprising a plurality of cache memories, each cache associated with a respective processor of said plurality of processors.

43. A multiprocessor server system as described in claim 36 wherein state information of any given packet of a same connection is preserved because said packets of said same connection are individually mutually excluded from said protocol layers.

44. A computer system comprising a processor coupled to a bus and a memory coupled to said bus and comprising instructions that when executed implement a method for processing data packets comprising:

accessing a packet associated with a connection; and

processing said packet through said plurality of protocol layers using a single thread from a single processor by assigning said connection for processing to a single processor of a multiprocessor server system wherein packets associated with said connection are directed to said single thread associated with said single processor for processing, wherein connection state information used by said plurality of protocol layers is preserved by mutual exclusion of other threads processing packets for said connection through said plurality of protocol layers.

45. The computer system as described in claim 44 wherein said single thread is uninterrupted while processing said packet through said plurality of protocol layers.

46. The computer system as described in claim 44 wherein said packet are assigned to a processing queue wherein said processing queue provides single threaded processing of said packet through said plurality of protocol layers.

47. The computer system as described in claim 46 wherein said processing queue provides single threaded processing of said packet through said plurality of protocol layers by assigning only one packet to be processed by said plurality of protocol layers at a time.

48. The computer system as described in claim 47 wherein said packet is assigned to said processing queue based on address information of said connection.

49. The computer system as described in claim 44 wherein a unique connection data structure is generated specific to said connection based on address information of said connection stored in said packets associated with the connection.

50. The computer system as described in claim 49 wherein said address information comprises a local IP address and a remote IP address.

51. The computer system as described in claim 49 wherein said address information further comprises a remote port address and a local port address.

53. A computer system comprising a processor coupled to a bus and a memory coupled to said bus and comprising instructions that when executed implement a method for processing data packets comprising:

accessing a packet associated with a connection; and

assigning said packet to a processing queue associated with a single processor of a multi processor server system, wherein said processing queue provides uninterrupted single threaded processing of said data packet associated with the connection through a plurality of protocol layers using a single thread of the single processor, wherein state information of the packet within the connection is preserved so as to mutually exclude other threads from processing packets of said connection through said plurality of protocol layers..

54. The computer system as described in claim 53 wherein said processing queue provides mutual exclusion of same-connection packet processing through said plurality of protocol layers.

56. The computer system as described in claim 53 wherein a unique connection structure associated with said connection is generated based on address information of said connection stored in the packets associated with the connection.

57. The computer system as described in claim 56 wherein said address information comprises a local IP address and a remote IP address.

58. The computer system as described in claim 57 wherein said address information further comprises a remote port address and a local port address.

59. The computer system as described in claim 58 wherein said connection structure is used to assign subsequent packets associated with said connection to said processing queue.

60. The computer system as described in claim 53 wherein said plurality of protocol layers includes a TCP protocol layer.

61. The computer system as described in claim 53 wherein said plurality of protocol layers includes an IP protocol layer.

IX. EVIDENCE APPENDIX

No copies of evidence are required with this Appeal Brief. Appellants have not relied upon any evidence submitted under 37 C.F.R. §§ 1.130, 1.131, or 1.132.

X. RELATED PROCEEDINGS APPENDIX

There are no copies of decisions rendered by a court or the Board to provide with this Appeal as there are no related proceedings.